Thread merging schemes for multithreaded clustered VLIW processors

Manoj Gupta* **Fermín Sánchez** **Josep Llosa**
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

Several multithreading techniques have been proposed to reduce the resource underutilization in Very Long Instruction Word (VLIW) processors. Simultaneous MultiThreading (SMT) is a popular technique which improves processor performance by issuing multiple instructions from different threads. SMT requires extra hardware to merge instructions from different threads. The complexity of this hardware increases substantially with the number of threads, limiting the number of threads that can be realistically supported to only 2. Cluster-level Simultaneous MultiThreading (CSMT) is a technique that merges instructions from threads at the cluster level. CSMT has a much lower merging hardware cost and can support a larger number of threads. However, CSMT performance is lower than SMT. In this paper, we evaluate several hardware designs that can support a high number of threads by using a merging scheme that combines both SMT and CSMT merging. For instance, one of the evaluated schemes, which merges the first 2 threads using SMT and the produced merging with other 2 threads by CSMT, achieves performance similar to supporting 4 threads by SMT but maintaining a reasonable merging hardware cost.

1 Introduction

Very Long Instruction Word (VLIW) processors have gained wide acceptance in the embedded domain due to their hardware simplicity, low cost and low power consumption [3, 9, 19]. To exploit high Instruction Level Parallelism (ILP), VLIWs need to be designed with a significant issue width. However, the centralized Register File (RF) becomes a bottleneck because of an increase in RF delay, power and area [18]. Clustered VLIW architectures have multiple RFs and cluster the Functional Units (FUs) to the RFs they are connected to. Many VLIWs have been designed using the clustered approach [9, 19].

Some applications scale well with issue width, which makes a high issue width processor desirable. However, the ILP exposed in many applications, or in some code regions, is limited and the processor is heavily underutilized. Besides, in a production environment, high ILP applications (like image processing) coexist with low ILP applications (like control code or the OS itself). In the context of VLIWs, processor underutilization can be characterized in terms of vertical and horizontal waste. Vertical waste are the cycles where no operations are issued at all. Horizontal waste is the underutilization of the issue width of the processor. Both vertical and horizontal waste arise because control and data dependencies in the program limit the number of operations that can be issued in a given cycle.

Several multithreading techniques have been proposed to reduce the vertical and horizontal waste in the processor. Block MultiThreading (BMT) [15, 24] executes instructions from a single thread until it is blocked by a long latency event (a cache miss, for instance). Interleaved MultiThreading (IMT) [17, 20] does a zero cycle context switch every cycle, so that instructions from different threads are interleaved at execution time. Simultaneous MultiThreading (SMT) [21] issues each cycle multiple instructions from multiple threads. In a SMT processor, issue-slots of the processor are filled by operations of different threads, converting thread level parallelism (TLP) into ILP. Several hybrid approaches combine some of these techniques: IMT & BMT [10], IMT & SMT [5], BMT & SMT [22].

To issue VLIW instructions from different threads simultaneously, SMT requires extra hardware to merge VLIW instructions from different threads (each instruction containing multiple operations) into a single execution packet. The obtained execution packet is issued as a normal VLIW instruction. The cost of the merging hardware for SMT increases substantially with the number of threads and limits its scalability. Cluster-level simultaneous MultiThreading (CSMT) [6] merges instructions at the cluster level and has a lower merging hardware cost. However, CSMT perfor-

*This work is supported by Spanish Ministry of Science and Technology under contract CICYT TIN2007-60625, FI grant from AGAUR/Generalitat de Catalunya, SARC (Scalable computer ARChitecture) Project and HiPEAC European Network of Excellence.

0190-3918/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPP.2009.48

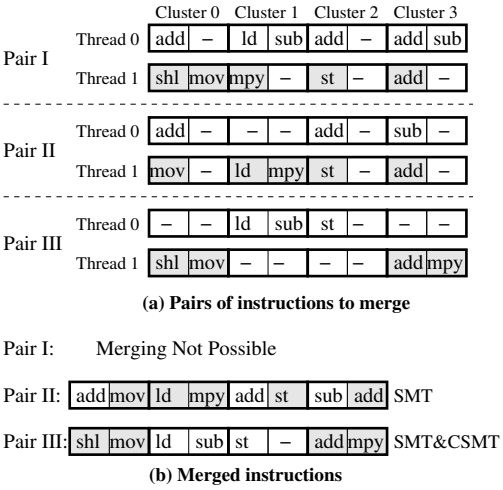

Figure 1: Instruction Merging in SMT and CSMT

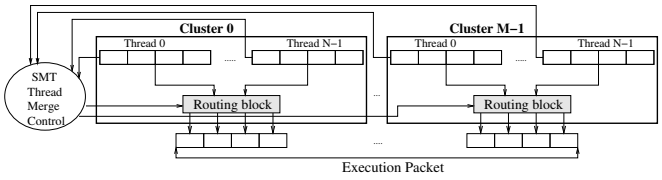

Figure 2: SMT Merging Hardware for a 4-issue per cluster Processor

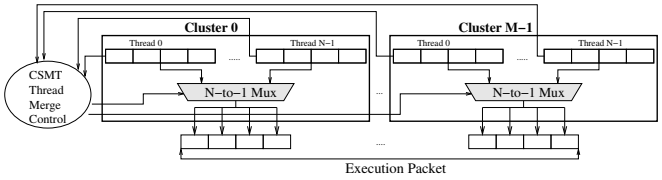

Figure 3: CSMT Merging Hardware for a 4-issue per cluster Processor

mance is lower than SMT.

In this paper, we propose several schemes for achieving the high SMT performance at a reasonable hardware cost. Many of these schemes use a mix of CSMT and SMT merging instead of using SMT only. Merging only few threads using SMT and rest by CSMT reduces the cost of merging hardware and makes it practical to support more threads.

The rest of the paper is organized as follows. Section 2 describes briefly the differences between CSMT and SMT and the corresponding merging hardware. Section 3 discusses the performance comparison and a cost analysis of CSMT and SMT. The various merging schemes and their cost evaluation are discussed in Section 4. A detailed performance of the merging schemes is presented in Section 5. Finally, Section 6 concludes the paper.

2 Background

2.1 SMT and CSMT Merging Example

In this section, we briefly describe the difference in merging threads between SMT and CSMT. In CSMT, the instruction merging is done at a cluster-level granularity instead of the operation-level merging done by SMT. Hence, CSMT issues instructions from multiple threads simultaneously only when the threads use different clusters. SMT, on the other hand, does not have this restriction and instructions from different threads may be simultaneously issued even if they share clusters. Note that when two instructions are merged, they have to be merged in their entirety, i.e. it is not possible to choose only a non-conflicting part of an instruction because doing so breaks VLIW execution semantics. An example of the merging done by SMT and CSMT is shown in Figure 1. Figure 1(a) displays 3 pairs of instructions for a 4-cluster 2-issue per cluster (8-issue) architecture. Each pair corresponds to one VLIW instruction for each thread that we intend to merge. Operations in the white background belong to Thread 0 and operations with a grey background belong to Thread 1. The final execution packet obtained by merging is shown in Figure 1(b). Neither CSMT nor SMT can merge Pair I because of conflicts at clusters 0, 1 and 3, both at operation-level and cluster-level. Pair II can be merged by SMT since there are no conflicts at operation-level. CSMT, however, cannot merge this pair, because there is a conflict at cluster-level at clusters 0, 2 and 3. As CSMT checks resource conflicts at cluster-level. Pair III, however, can be merged by CSMT (and SMT as well) as the first instruction uses only clusters 1 and 2 which are not used by the other instruction. As shown in the example, using CSMT restricts the opportunities to merge the instructions in comparison to SMT. However, doing so results into a lower cost merging hardware for CSMT. The following section discusses the merging hardware for SMT and CSMT in more detail.

2.2 SMT and CSMT Merging Hardware

Figure 2 shows the implementation of the SMT merging hardware for a N-thread M-cluster processor with a 4-issue per cluster. The SMT merging hardware consists of a thread merge control and a routing block per cluster. SMT thread merge control checks resource collisions[1] at operation-level and selects the threads that can be issued simultaneously at a given cycle. To fit operations from multiple threads in the same cluster, the operations of the instructions may need to

[1] In most VLIW processors, certain type of operations can be executed only at fixed issue slots. For instance, in our base architecture, while ALU operations may be executed at any issue slot, operations like memory load/store, multiply and branch can only be executed at their fixed slots.

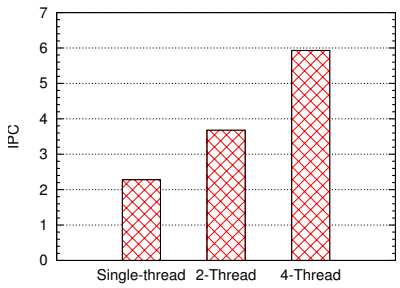

Figure 4: SMT Performance

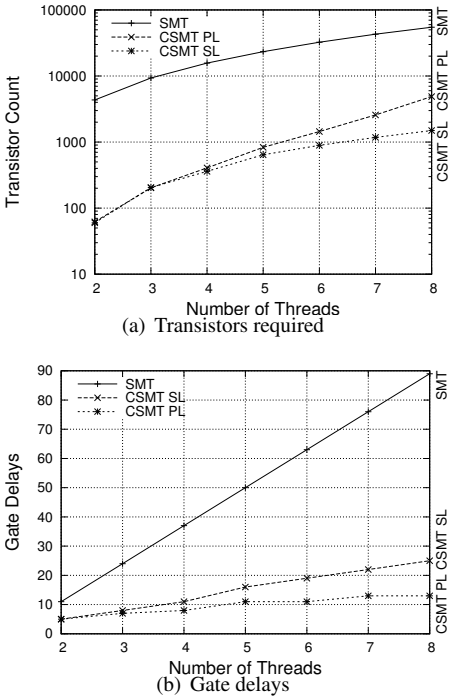

Figure 5: CSMT and SMT thread merge control cost

be rerouted. SMT thread merge control generates the appropriate signals for routing the operations. The routing block uses these signals to produce the final execution packet.

The merging hardware for CSMT is shown in Figure 3. CSMT merging hardware consists of a thread merge control and a multiplexer per cluster. Thread merge control checks for resource collisions at cluster-level and selects the threads that can be issued simultaneously. The multiplexers select the cluster from the threads selected by the thread merge control. Note that no routing of operations is required for CSMT because operations from only one thread can be issued at a given cluster. Thus, CSMT thread merge control has a lower cost than SMT. However, the area required by the multiplexers is similar to the area required for the routing block in SMT, according to the methodology used in [12] for interconnect area computation. This is

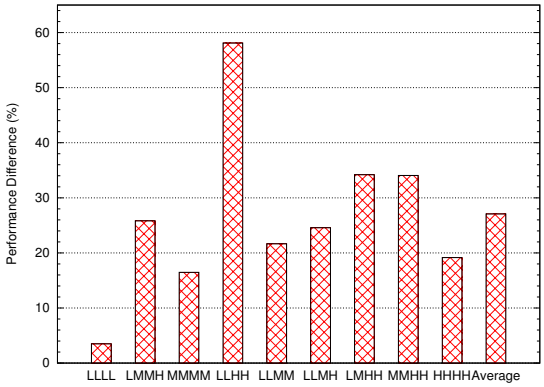

Figure 6: SMT performance advantage over CSMT

because the number of input and output wires for the routing block is the same as that of the multiplexers. Then for both CSMT and SMT, the cost of the thread merge control is the only variable cost in the merging hardware (the multiplexers are indispensable even for the simple multithreading scheme IMT). Hence, the cost of the thread merge control is the only differentiating factor between CSMT and SMT.

3 Motivation

Supporting a large number of threads on a SMT processor is desirable because of the performance gains that are achieved. To illustrate this, Figure 4 shows the average IPC obtained by SMT for the workloads evaluated in this paper (the experimental setup and the workloads are explained later in Section 5.1) on a single-thread, 2-thread and a 4-thread processor. SMT performance improves significantly with the number of threads and the 4-Thread SMT processor has a performance advantage of 61% over a 2-Thread SMT Processor. However, the hardware cost increases considerably with the number of threads. The increased cost limits the number of threads that can be realistically supported by SMT to only 2.

Figure 5 shows the cost of the thread merge control hardware with varying number of threads for a 4-cluster 4-issue per cluster architecture for both CSMT and SMT. Figure 5(a) shows on a logscale the cost in terms of number of transistors required, and Figure 5(b) shows the cost in terms of gate delays. The values for CSMT have been taken from [7]. The computation details of the gate delays and transistor count for SMT thread merge control are computed following the same methodology as in [7] and are omitted from this paper for space reasons. Two implementations of thread merge control are considered for CSMT [7] in this paper viz. serial and parallel. In the figures, labels 'CSMT PL' and 'CSMT SL' refer to the parallel and serial implementations of CSMT thread merge control, while label 'SMT' refers to the SMT thread merge control. The serial imple-

mentation is a cascading logic checking a different thread at each level. The parallel implementation, on the other hand, checks, in parallel, all possible thread selections. The parallel implementation has a lower delay and is functionally equivalent to the serial implementation. However, parallel implementation has a higher hardware overhead, which grows exponentially with the number of threads.

For SMT thread merge control, an implementation similar to the 'CSMT SL' approach is considered. The parallel approach is not feasible for SMT because the cost of checking, in parallel, all possible thread selections is prohibitively expensive in terms of area. As shown in the figures 5(a) and 5(b), the cost of the thread merge control for SMT increases significantly with the number of threads and constrain its scalability. In fact, previous studies that have done an evaluation of the hardware required for SMT on VLIW [16] also limit the number of threads that can be realistically supported to only 2.

Compared to SMT, CSMT scales better with number of threads and 4 threads can be supported. However, CSMT has a lower performance than SMT, diminishing its attractiveness. Figure 6 shows the difference in performance between SMT and CSMT for a 4-Thread processor for the workloads used in this paper (explained later in Section 5.1). On an average, SMT performance is 27% higher than CSMT. For particular cases like LLHH (2 benchmarks with low ILP and 2 benchmarks with high ILP), the performance difference is as high as 58%.

The cost of CSMT merge control is much lower compared to SMT merge control. Thus, to maintain high performance by supporting more threads while keeping the cost of merging hardware low, we intend to combile both SMT and CSMT in the merge control hardware. For instance, the first two threads may be merged using SMT and the result is merged with another thread using CSMT. Supporting the extra threads using CSMT has little impact on the total cost of the merging hardware and improves the performance. Following section discusses the merging schemes we have considered in detail.

4 Merging Schemes Exploration

This section explores various merging schemes that can be deployed to support more than 2 threads at low cost. For space reasons, we limit our evaluations in this paper to a 4-Thread architecture only.

4.1 Merging Schemes

Figure 7(a) shows the merging scheme used for SMT. In the scheme, the first two threads (T0 and T1) are merged, the result is merged with the next Thread T2 and so on, using operation-level merging hardware. CSMT merge control has a much lower cost than SMT. This provides an opportunity to support more threads at reasonable cost and achieve higher performance at low cost by combining both CSMT and SMT approaches for the merge control. Figure 7(b) shows an example of a merging scheme combining both SMT and CSMT. In the example, the first two threads (T0 and T1) are merged at operation-level (SMT) but the result and the next two threads (T2 and T3) are merged at cluster-level (CSMT). Since the cost of CSMT merging is much lower than SMT, the addition of CSMT has little impact on the overall cost of the merging hardware. Thus, the cost is comparable to a 2-Thread SMT merging even though 4 threads are simultaneously supported and merged. Figures 8(a)-8(h) show all the possible schemes in which SMT and CSMT merge control are combined for a 4-Thread architecture. In all the schemes shown, the first digit refers to the number of levels of cascade, and each following letter indicates whether the merging at each level is CSMT ('C') or SMT ('S'). For instance, scheme 3SCC implies that there are 3 levels of cascade, with SMT merging at the first level and CSMT merging at the second and the last level.

Note that if more than two threads are merged by CSMT, two different implementations, serial and parallel, are possible. Figures 8(a), 8(c) and 8(d) show the serial implentations and the parallel implementations are shown in figures 8(i)-8(k). Thus, merging 4 threads by using CSMT SL corresponds to Figure 8(a) while merging 4 threads using CSMT PL is shown in Figure 8(k). Use of the CSMT parallel implementations is indicated by a subscript showing the number of threads being merged in parallel. For instance, C_4 refers to merging 4 threads using the parallel approach. The parallel implementation is also possible for SMT. However, the cost of checking multiple threads in parallel is prohibitively expensive in terms of area. For this reason, parallel implementations for SMT for more than 2 threads are not considered.

The merging schemes explored in figures 8(a) to 8(k) use a cascade to merge the threads i.e. initially, the first two threads are merged, then the result is merged with the next thread and so on. We also explore a different scheme for merging the threads which is analogous to a balanced tree structure and lowers the delay of the merging hardware. In these schemes, the 4 threads are divided into groups of 2 threads each. Threads in each group are first merged independently of the other group. The obtained mergings for the two groups are then merged producing the final execution packet. Figure 8(l) shows an example of this approach where threads (T0,T1) and (T2,T3) are merged with CSMT and the two resulting merges are merged again using CSMT. This approach results into a lower delay than the one presented in Figure 8(a) because of the reduced number of levels of merging. These schemes are shown in figures 8(l)-8(o). The schemes 2SS and 2CC use either SMT or CSMT

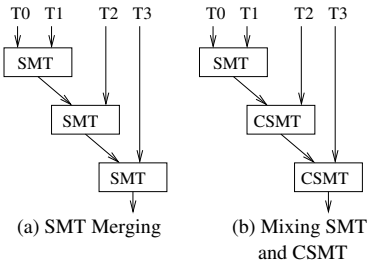

Figure 7: Mixed merging example

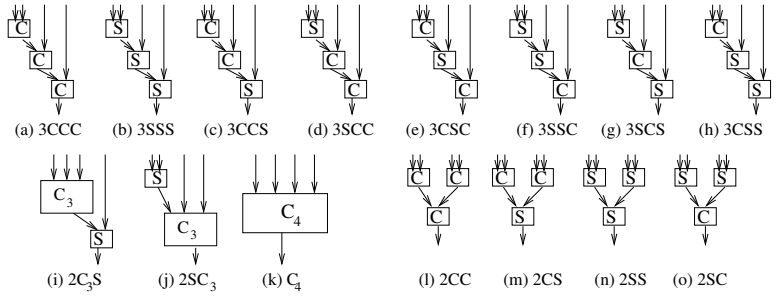

Figure 8: Possible merging schemes for 4 threads

merging only and can be considered as alternative implementations for a 4-Thread CSMT or SMT processor. Note that while these approaches lower the delay of the merging hardware, there might be an impact on performance. This is because merging T2 and T3 creates an instruction larger than the T2 or T3 individual instructions. It may happen that this larger instruction can not be merged with the merging produced by T0 and T1 even though T2 or T3 could be individually merged.

4.2 Cost Analysis

This section presents the cost analysis of the merging hardware for all the schemes considered in Section 4 in terms of gate delays and transistor count. Figure 9 shows the gate delays and the number of transistors required for each scheme. In the figure, the bars show the gate delays and the line shows the transistors required for each scheme. For the sake of comparison, the figure also includes the cost of the merging hardware of a 2-Thread SMT (1S). In general, the number of transistors required by any scheme is dominated by the number of SMT merge control blocks used by the scheme. Schemes that use only CSMT merging (C_4, 2CC and 3CCC) are the cheapest overall. Amongst the schemes that use SMT, schemes with only 1 SMT merge control block (1S, 3SCC etc.) are the least expensive, while schemes with 3 SMT merge control blocks (2SS and 3SSS) are the most expensive. As expected, there is little difference in the transistor requirement of a 2-Thread SMT (1S) and the schemes that use only 1 SMT merge control block because the cost of CSMT merge control is an insignificant addition to the total cost.

Schemes that use only CSMT (C_4, 2CC and 3CCC) also have the lowest gate delays, highlighting the advantages of CSMT over SMT. Schemes that use SMT have higher delays. Note that the gate delays of the schemes $2SC_3$, 3SCC and 2SC are very close to the gate delays for a 2-Thread SMT (1S). Schemes $2SC_3$ and 3SCC also have a transistor requirement similar to 1S, making these schemes particularly attractive for implementation. Other schemes have much higher gate delays. Also note that the schemes 3CSC

Figure 9: Merging hardware cost

and 3CCS have higher gate delays than 3SCC or $2SC_3$ even though all these schemes use a single SMT merge control block. This is because in 3SCC and $2SC_3$, using SMT merge control earlier during merging allows the computation of routing of the operations in a VLIW instruction to be done in parallel with CSMT merging. Parallel computation of the routing also results into the lowest delay for scheme 3SSC compared to similar schemes 3SCS and 3CSS.

5 Performance Evaluation

5.1 Experimental Setup

The evaluation done in this paper is based on the VEX clustered architecture [23] modeled upon the commercial HP/ST Lx [3] VLIW family. The VEX C compiler [23] used in this study is a derivation of the HP/ST ST200 C compiler, which itself is a derivative of the Multiflow compiler [14] that uses *Trace Scheduling* [4] as global scheduling algorithm and *Bottom Up Greedy* [2] as cluster assignment algorithm. Each cluster has 2 multipliers and 1 load/store unit, and the number of ALUs is the same as the issue width of the cluster (4 in our experiments). Memory

Table 1: Benchmarks

Benchmarks	*ILP Degree*	*Description*	IPC_r	IPC_p
mcf	L	Minimum Cost Flow	0.96	1.34
bzip2	L	Bzip2 Compression	0.81	0.83
blowfish	L	Encryption	1.11	1.47
gsmencode	L	GSM Encoder	1.07	1.07
g721encode	M	G721 Encoder	1.75	1.76
g721decode	M	G721 Decoder	1.75	1.76
cjpeg	M	Jpeg Encoder	1.12	1.66
djpeg	M	Jpeg Decoder	1.76	1.77
imgpipe	H	Imaging pipeline	3.81	4.05
x264	H	H.264 encoder	3.89	4.04
idct	H	Inverse Discrete Cosine Transform	4.79	5.27
colorspace	H	Colorspace Conversion	5.47	8.88

Table 2: Workload configurations

ILP Comb	*Thread 0*	*Thread 1*	*Thread 2*	*Thread 3*
LLLL	mcf	bzip2	blowfish	gsmencode
LMMH	bzip2	cjpeg	djpeg	imgpipe
MMMM	g721encode	g721decode	cjpeg	djpeg
LLMM	gsmencode	blowfish	g721encode	djpeg
LLMH	mcf	blowfish	cjpeg	x264
LLHH	mcf	blowfish	x264	idct
LMHH	gsmencode	g721encode	imgpipe	colorspace
MMHH	djpeg	g721decode	idct	colorspace
HHHH	x264	idct	imgpipe	colorspace

and multiply operations have a latency of 2 cycles, and the rest have single-cycle latency. There is no branch predictor and fall-through path is the predicted path. The incorrect instructions issued following a taken branch are squashed. Assuming a dedicated pipeline stage for thread merging, the taken branch penalty is 2 cycles.

Experiments have been done in a 16-issue, 4-cluster architecture configuration (i.e. 4-issue per cluster). All the experiments have been done assuming a 64KB, 4-way set-associative, 20-cycles miss penalty design for both ICache and DCache (assuming a processor frequency[2] of 400 MHz. and a worst case DRAM latency of 50 ns for critical word transfer). We have used a set of MediaBench [13] and SpecInt 2000 [8] applications. We have also included production color space conversion [1], imaging pipeline [23] used in high performance printers, inverse discrete cosine transform (used in various codecs) [11] and H.264 encoder [25]. The benchmarks are shown in Table 1. Column IPC_r shows the average IPC of the first 100 million VLIW instructions (the number of instructions executed for each thread in our experiments) for each benchmark. Column IPC_p shows the average IPC of the first 100 million VLIW instructions for a perfect memory without cache misses. Benchmarks are classified by their IPC_p in three categories: high IPC (idct, x264, colorspace and imgpipe), medium IPC (g721encode, g721decode, cjpeg and djpeg) and low IPC (mcf, bzip2, blowfish and gsmencode). This classification is shown in column $ILP\ Degree$ as L (low IPC), M (medium IPC) and H (high IPC) with 4 benchmarks lying in each category.

[2]Frequency of the fastest processor, ST231, in ST200 family.

The workload configurations used for evaluation are listed in Table 2. In order to select appropriate thread configurations, we have combined benchmarks with different ILP degrees, attempting to cover representative combinations. Column labeled as $ILP\ Comb$ indicates these combinations. For example, configuration LLHH in Table 2 has two benchmarks with low IPC and two benchmarks with high IPC, configuration LLMM has two benchmarks with low IPC and two benchmarks with medium IPC and configuration LMHH has one benchmark with low IPC, one benchmark with medium IPC and two benchmarks with high IPC. We carried out the experiments by arranging the workloads in a multitasking environment. The number of threads supported by processor is exposed as virtual CPUs and the operating system schedules as many threads to run as the number of virtual CPUs, with a timeslice of 1 million cycles. After the expiry of the timeslice, a context switch takes place and the running threads are replaced by other threads from the workload. For a single-thread processor, the threads run in serial order with a single thread running in the whole timeslice. For a 2-thread processor, 2 threads are scheduled to run together in the same timeslice and, for a 4-thread processor, 4 threads share the timeslice. To improve fairness and to alleviate any bias, replacement threads are picked at random from the workload after the context switch. The workloads are executed till one thread completes executing 100 million VLIW instructions.

5.2 Results

This section presents the performance results obtained for the merging schemes evaluated in this paper. Figure 10 shows the performance obtained for all the schemes evaluated and also a 2-Thread SMT (1S) configuration. Schemes 3CCC and C_4 are two different implementations of a 4-Thread CSMT configuration, while scheme 3SSS is the 4-Thread SMT configuration and achieves the peak performance. We found that the schemes (3CCC,C_4), (3SCC,3CSC,3CCS,2SC_3,2C_3S) and (3CSS,3SCS,3SSC) obtained very similar performance (less than 1% difference in performance for all workloads. Besides, several schemes like (3SCC,2SC_3) and (3CCC,C_4) etc. are identical in terms of performance). Hence, these schemes have been grouped together in the figures.

Amongst all the schemes (except 1S and 3SSS, that represent the minimum and the maximum performance), schemes (3CSS,3SCS,3SSC) performs the best with a performance within 5.6% of the peak 3SSS performance. On the other hand, the scheme 2SC performs the worst and achieves a performance even lower than 3CCC (and only marginally better than 1S) despite having a much higher cost. This is because using CSMT merging after the threads have been merged using SMT results into a sig-

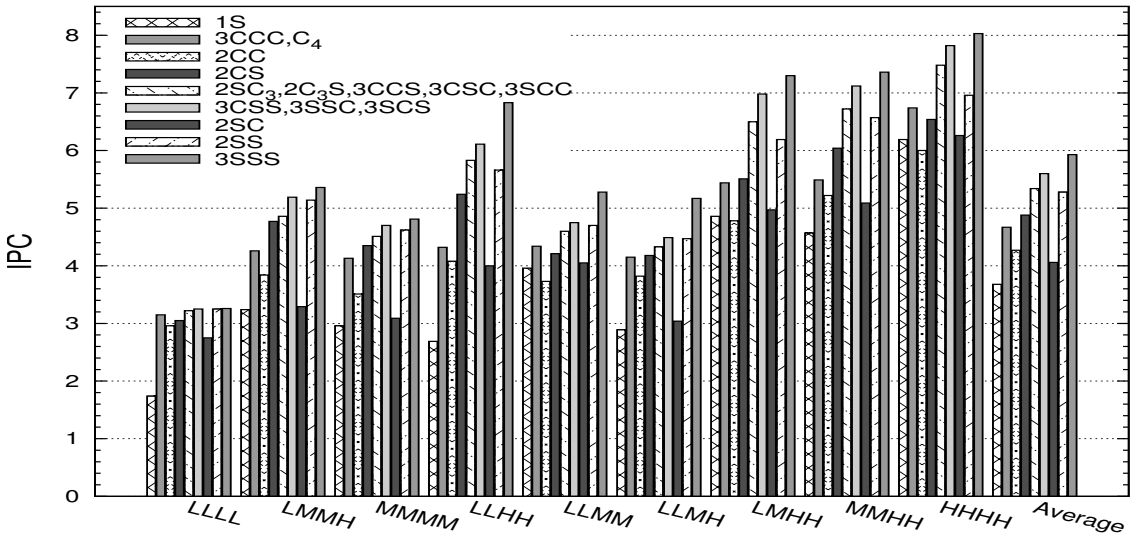

Figure 10: Merging schemes performance

nificant restriction on merging. Scheme 2CC also has a lower performance than 3CCC albeit a little better than 2SC. Next we evaluate the the schemes where only 1 SMT merge control block is used in the merging hardware, i.e. the schemes 3SCC, 3CCS, 2CS, $2C_3S$, $2SC_3$ and 3CSC. In general, scheme 2CS performs the worst among all the schemes with a single SMT merge control block and achieves a performance only marginally higher than 3CCC. Other schemes with a single SMT merge control block, (3SCC,3CSC,3CCS,$2SC_3$,$2C_3S$), have a performance lower than the schemes (3CSS,3SCS,3SSC). However, they still significantly outperform a 4-Thread CSMT (14%) and a 2-Thread SMT processor (45%). Also, the performance is only 11% lower than the 4-Thread SMT processor (3SSS) on an average. Scheme 2SS also has a performance comparable to schemes (3SCC,3CSC,3CCS,$2SC_3$,$2C_3S$).

We now present an analysis of the performance of the merging schemes while taking their cost into account as well. Figures 11 and 12 show the performance of the merging schemes in combination with the required transistors and the gate delays respectively. In the figure 11, schemes 3SCC,3CSC,3CCS,$2SC_3$,$2C_3S$ and schemes 3CSS,3SCS,3SSC have been grouped together as they have similar results for performance and transistors incurred.

As shown in both figures 11 and 12, schemes that use only CSMT merging (3CCC, C_4 and 2CC) are the cheapest in terms of both delay and transistors incurred. These schemes are the only choice if the design is quite constrained and even the cost of a 2-Thread SMT can not be supported. However, scheme 2CC is not an attractive choice because of its lower performance compared to schemes 3CCC and C_4, that have higher performance but similar

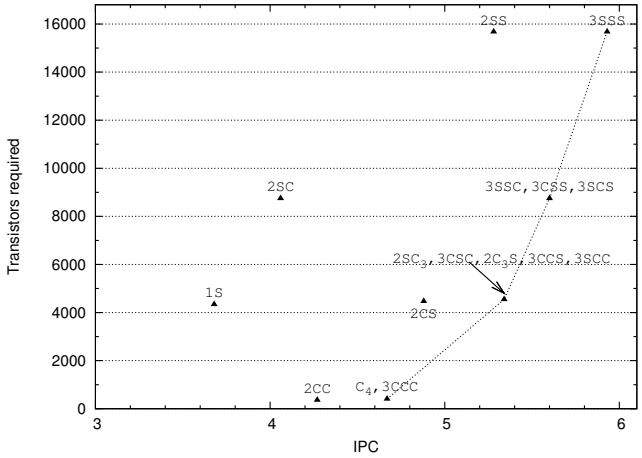

Figure 11: Performance vs transistors incurred

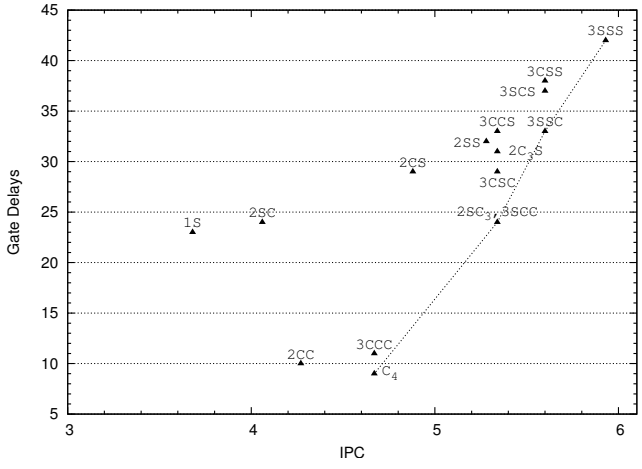

Figure 12: Performance vs gate delays

cost. If the cost of a 2-Thread SMT can be afforded, then schemes $2SC_3$ and 3SCC are attractive as they have a higher performance than a 2-Thread SMT but at a similar cost both in terms of area and gate delays. These schemes even outperform the more complex schemes 2SS, 2SC and 2CS.

Only the schemes 3SSC, $3SC_3$ and 3SCS have a higher performance than $2SC_3$. Among these schemes, 3SSC is the best choice because of its lower delay compared to others with similar performance and transistor requirement. However, the extra performance between 3SSC and $3SC_3$ comes at the cost of supporting 2 SMT merge control blocks. Besides, the performance difference is not significant enough to justify the additional cost.

In conclusion, for all the schemes evaluated, scheme $2SC_3$ presents a good tradeoff between performance and cost because of its low cost (comparable to a 2-Thread SMT) and a performance close to a 4-Thread SMT.

6 Conclusions

Simultaneous MultiThreading (SMT) is a popular approach for improving processor performance. However, SMT is expensive to scale beyond 2 threads because of the increased cost of the merging hardware. Other schemes like Cluster-level Simultaneous MultiThreading (CSMT) has a lower merging hardware cost and can support higher number of threads. However, CSMT performance is lower than SMT.

This paper explored several merging hardware designs which use a combination of both CSMT and SMT merging. The use of CSMT in the merging hardware allows supporting more threads without siginificantly affecting the cost of the merging hardware. The experimental results prove that the performance of the combined merging is quite reasonable and achieves a performance in-between the two extremes CSMT and SMT. One of the schemes, $2SC_3$ in particular, which uses SMT to merge 2 threads and merges the rest using CSMT, is quite attractive. $2SC_3$ merges 4 threads, has a cost close to the 2-Thread SMT merging but has a much higher performance. On an average, $2SC_3$ achieves 14% higher performance than a 4-Thread CSMT processor, 45% higher performance than a 2-Thread SMT processor and is within 11% of a 4-Thread SMT processor performance.

References

[1] Colorspace Conversion Program Used in High Performance Printers, Personal Communication.

[2] J. R. Ellis. *Bulldog: a compiler for VLSI architectures*. MIT Press, Cambridge, MA, USA, 1986.

[3] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood. Lx: A Technology Platform for Customizable VLIW Embedded Processing. In *ISCA*, 2000.

[4] J. A. Fisher. Trace Scheduling: A Technique for Global Microcode Compaction. *IEEE Trans. Computers*, 30(7):478–490, 1981.

[5] M. Gupta, F. Sánchez, and J. Llosa. Hybrid Multithreading for VLIW Processors. Technical Report UPC-DAC-RR-CAP-2009-19.

[6] M. Gupta, F. Sánchez, and J. Llosa. Cluster-Level Simultaneous MultiThreading for VLIW Processors. In *ICCD*, 2007.

[7] M. Gupta, F. Sánchez, and J. Llosa. Merge Logic for Clustered Multithreaded VLIW Processors. In *EUROMICRO Conference on Digital System Design*, 2007.

[8] J. L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Computer*, 33(7):28–35, 2000.

[9] F. Homewood and P. Faraboschi. ST200: A VLIW Architecture for Media-Oriented Applications. *Microprocessor Forum*, 2000.

[10] J. Hoogerbrugge and A. Terechko. A multithreaded multicore system for embedded media processing. *Transactions on HiPEAC*, 3(2), 2008.

[11] Inverse discrete cosine transform, taken from ffmpeg. www.ffmpeg.org. last consult may 2009.

[12] R. Kumar, N. Jouppi, and D. Tullsen. Conjoined-Core Chip Multiprocessing. In *MICRO*, 2004.

[13] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *MICRO*, 1997.

[14] P. G. Lowney, S. M. Freudenberger, T. J. Karzes, W. D. Lichtenstein, R. P. Nix, J. S. O'Donnell, and J. C. Ruttenberg. The Multiflow Trace Scheduling Compiler. *The Journal of Supercomputing*, 7(1-2):51–142, 1993.

[15] A. Mikschl and W. Damm. MSparc: A Multithreaded Sparc. In *Euro-Par, Vol. II*, pages 461–469, 1996.

[16] E. Ozer and T. Conte. High-performance and low-cost dual-thread VLIW processor using Weld architecture paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1132–1142, 2005.

[17] R. Alverson et al. The Tera computer system. In *ICS*, 1990.

[18] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register Organization for Media Processing. In *HPCA*, pages 375–386, 2000.

[19] N. Seshan. High VelociTI Processing. *IEEE Signal Processing Magazine*, 15(2):86–101, March 1998.

[20] B. J. Smith. Architecture and Applications of the HEP Multiprocessor Computer System. In *SPIE*, 1981.

[21] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *ISCA*, pages 392–403, 1995.

[22] E. Tune, R. Kumar, D. Tullsen, and B. Calder. Balanced Multithreading: Increasing Throughput via a Low Cost Multithreading Hierarchy. In *MICRO*, 2004.

[23] VEX Toolchain. http://www.hpl.hp.com/downloads/vex/.

[24] W.-D. Weber and A. Gupta. Exploring the benefits of multiple hardware contexts in a multiprocessor architecture: preliminary results. In *ISCA*, pages 273–280, 1989.

[25] x264 - a free h264/avc encoder. http://www.videolan.org/developers/x264.html. Last consult May 2009.